FIG. 1

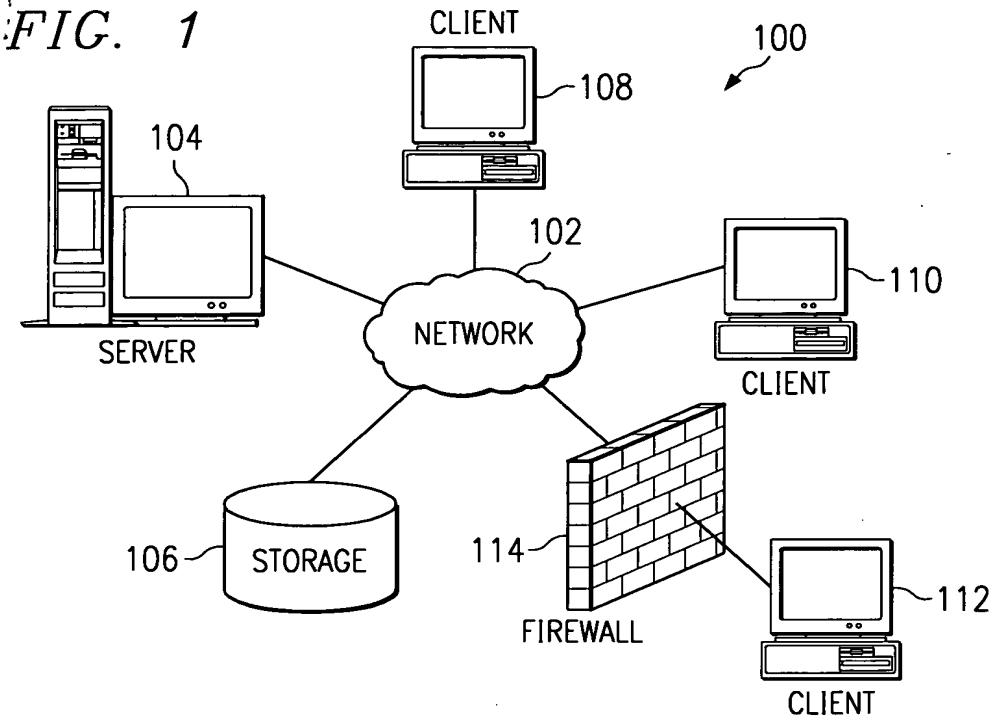
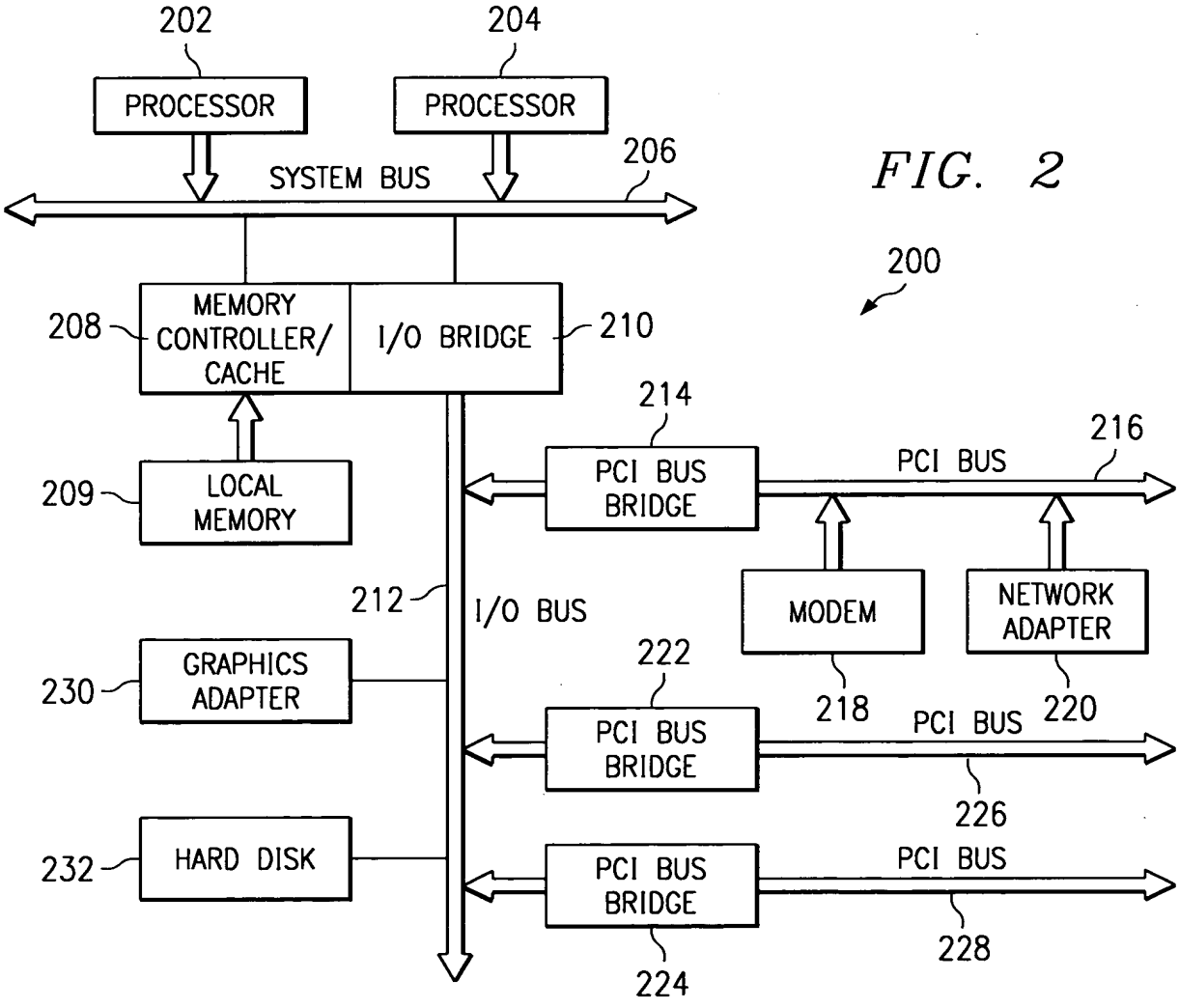

CLIENT
100
108
104
102
110
SERVER
NETWORK
CLIENT
106
STORAGE
114
FIREWALL
112
CLIENT

FIG. 2


202
204
PROCESSOR
PROCESSOR
206
SYSTEM BUS
200
208
MEMORY CONTROLLER/ CACHE
I/O BRIDGE
210
214
216
PCI BUS
209
LOCAL MEMORY
PCI BUS BRIDGE
212
I/O BUS
MODEM
NETWORK ADAPTER
222
230
GRAPHICS ADAPTER
218
PCI BUS
220
PCI BUS BRIDGE
226
232
HARD DISK
PCI BUS
PCI BUS BRIDGE
228
224

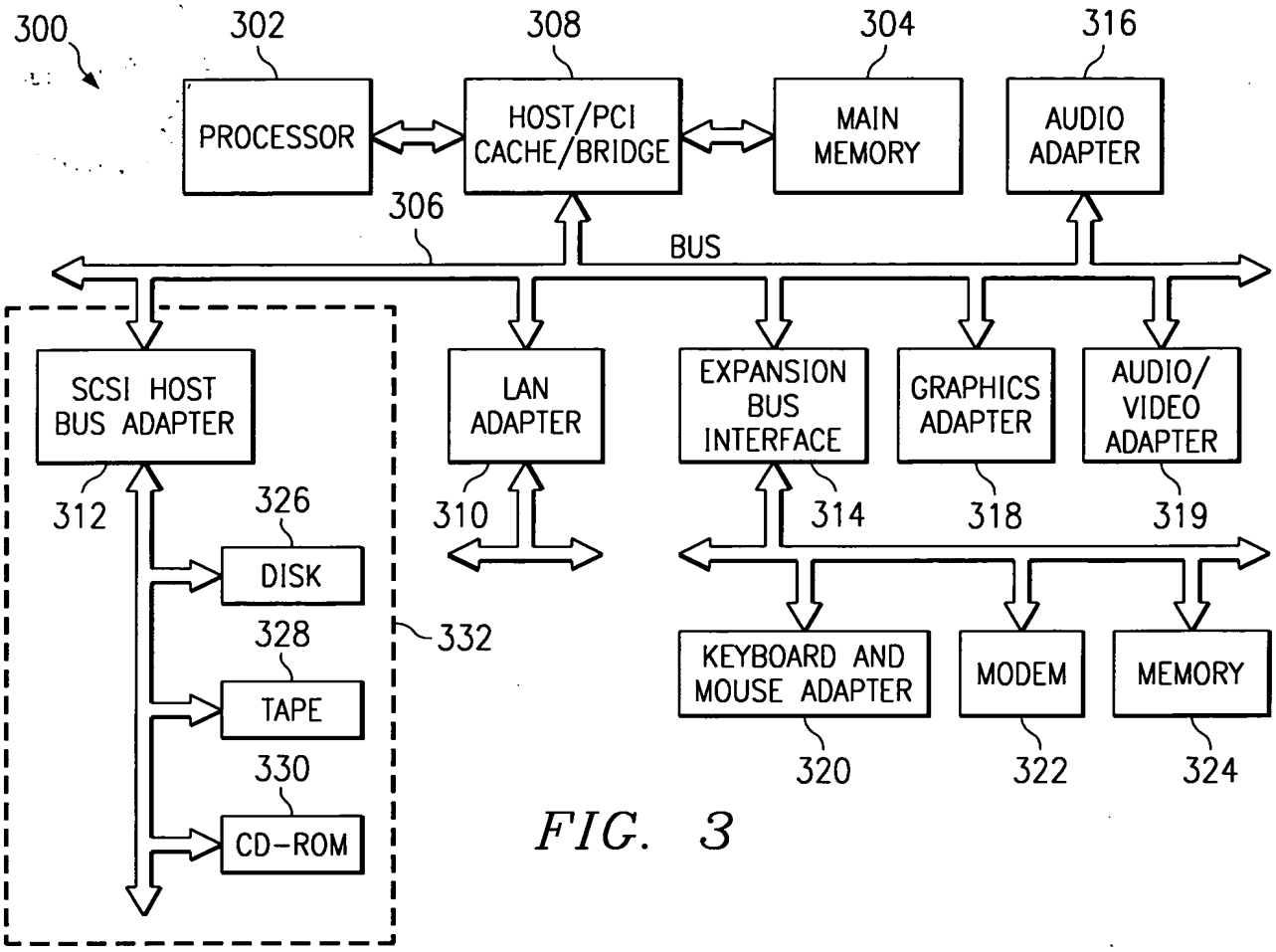

300
302
308
304
316
PROCESSOR
HOST/PCI CACHE/BRIDGE
MAIN MEMORY
AUDIO ADAPTER
306
BUS
SCSI HOST BUS ADAPTER
LAN ADAPTER
EXPANSION BUS INTERFACE
GRAPHICS ADAPTER
AUDIO/ VIDEO ADAPTER
312
310
314
318
319
326
DISK
328
TAPE
330
CD-ROM
332
KEYBOARD AND MOUSE ADAPTER
MODEM
MEMORY
320
322
324

FIG. 3

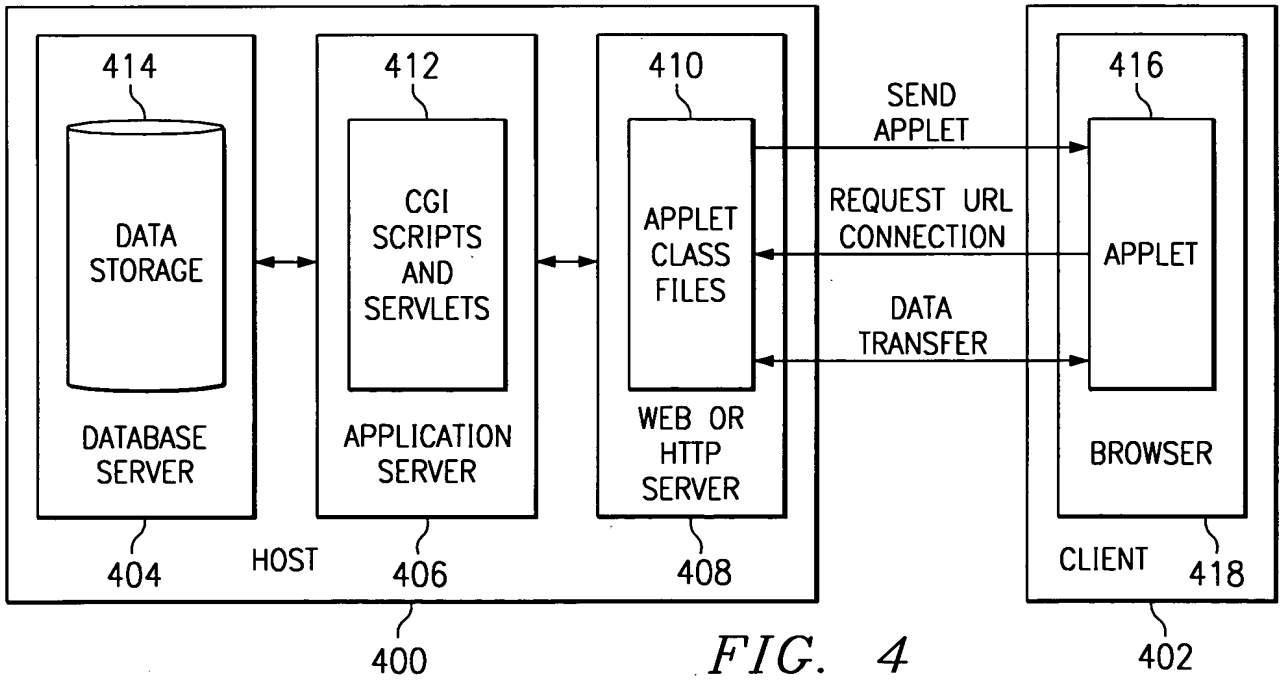

414
412
410
416
DATA STORAGE
CGI SCRIPTS AND SERVLETS
APPLET CLASS FILES
SEND APPLET
REQUEST URL CONNECTION
DATA TRANSFER
APPLET
DATABASE SERVER
APPLICATION SERVER
WEB OR HTTP SERVER
BROWSER
404
HOST
406
408
CLIENT
418
400
402

FIG. 4

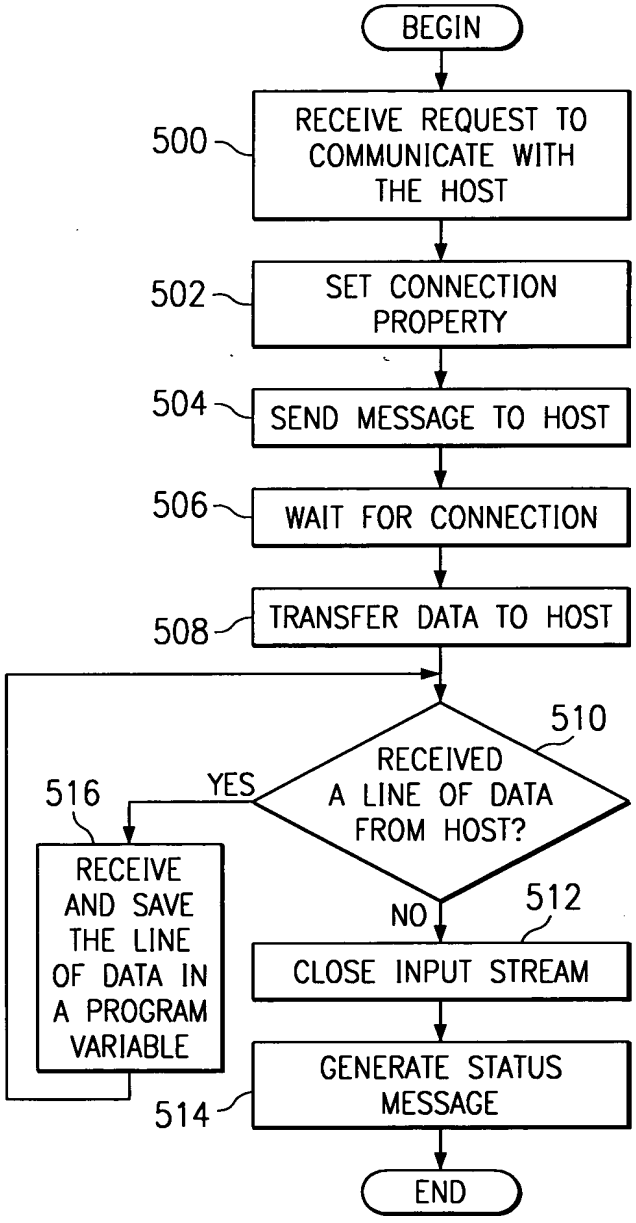

BEGIN
500
RECEIVE REQUEST TO COMMUNICATE WITH THE HOST
502
SET CONNECTION PROPERTY
504
SEND MESSAGE TO HOST
506
WAIT FOR CONNECTION
508
TRANSFER DATA TO HOST
510
RECEIVED A LINE OF DATA FROM HOST?
YES
516
RECEIVE AND SAVE THE LINE OF DATA IN A PROGRAM VARIABLE
NO
512
CLOSE INPUT STREAM
GENERATE STATUS MESSAGE
514
END

FIG. 5

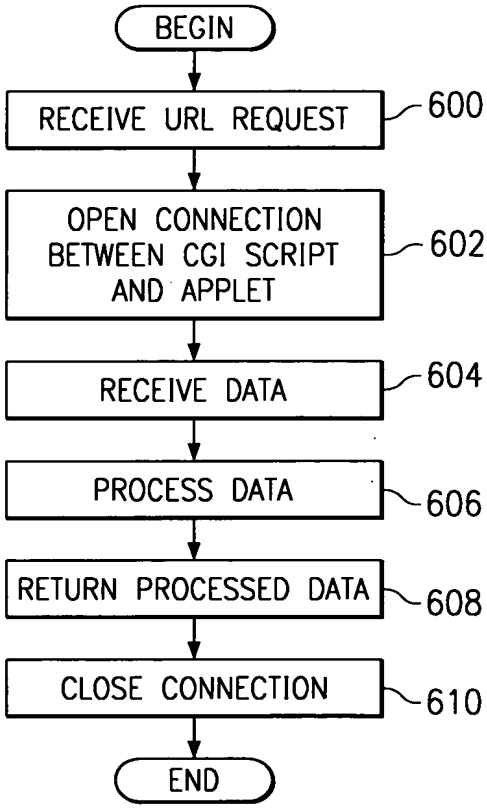

BEGIN
RECEIVE URL REQUEST
600
OPEN CONNECTION BETWEEN CGI SCRIPT AND APPLET
602
RECEIVE DATA
604
PROCESS DATA
606
RETURN PROCESSED DATA
608
CLOSE CONNECTION
610
END

FIG. 6

700

```
/*
 * This is a function that is executed by the applet. This Java code
 *       allows the applet to make a connection and then
 *       talk (exchange data) with the host that it was downloaded from.
 *
 * So, create a URL Connection to the server that you (applet) were
 *       launched from and then exchange data with the server.
 *
 * Inputs to this function are:
 * 1. aFileAtURL - This is the name of a servlet or CGI program at
 *                 the server.  The applet uses a servlet or CGI
 *                 helper program back at the host to handle Add,
 *                 Delete, Update and Retrieve of data from a
 *                 backend database.
 *
 * 2. dataString - This is the data stream that will be sent to the
 *                 host once a successful connection is created.
 *
 */
public void stdioTalkToHome (String aFileAtURL, String dataString )
{
    // The following 3 vars will be defined before the static {}
    //    block in the applet
    URL redirectURL = null;
    static final int DEFAULT_PORT = 80;
    String msgOutString    = "";

    URLConnection urlc = null;
    String host;
    int port;
    BufferedReader bri = null;
    DataOutputStream dos = null;
```

702

FIG. 7A

```
    // if a msg to the user is already showing, re-surface that msg
    if (getMsgHandle() != null)
    {
        getMsgHandle() .changeText("Connecting to host... Please Wait.") ;
        getMsgHandle() .showIt() ;
}
else
{
        ivMsg = new PopUpMsg( "Applet Status",
                              "Connecting to host... Please Wait.") ;
}

// find the host & port that I (applet) was launched from...
host = getDocumentBase() .getHost() ;
port = getDocumentBase() .getPort() ;

consoleDebug("host = " + host) ;
consoleDebug("port = " + port) ;

// if port is not set...
if (-1 == port)
{
        // force it to default port...
```

702

704

FIG. 7B

```
// Create the fully qualified URL string...
try
{
   redirectURL = new URL( getDocumentBase().getProtocol(),
                  host,
                  port,
                  aFileAtURL );
}
catch (MalformedURLException e)
{
   showHostCommError(1000, "Host URL could not be located for network communication.");
   showError(e);
   return;
}
consoleDebug("Home URL=" + redirectURL);

// Now, create the URLconnection to the host...
try
{
   urlc = redirectURL.openConnection();

   // next, change the setup parameters...
   urlc.setDoInput(true);
   urlc.setDoOutput(true);
   urlc.setUseCaches(false);
   urlc.setAllowUserInteraction(false);

   // and also the request property...
   urlc.setRequestProperty( "Content-Type", "application/x-www-form-urlencoded" );
}
```

706

708

FIG. 7C

```
catch (IOException e)
{
   showHostCommError(1001, "A URLConnection for host communication could not be created.");
   showError(e);
   return;
}
consoleDebug("Home URLConnection was created");


if (getMsgHandle() != null)
{
   getMsgHandle().changeText("Sending data to host... Please Wait.");
   getMsgHandle().showIt();
}

// Send the data from applet to the host server
try
{
   dos = new DataOutputStream(urlc.getOutputStream());
   dos.writeBytes(dataString);
   dos.close();
   consoleDebug("line written: " + dataString);
}
catch (IOException e)
{
   showHostCommError(1005, "User Data write failure occurred during host communication.");
   showError(e);
   return;
}
consoleDebug("All data was sent to host.");
```

708

710

FIG. 7D

```
// Create a Reader
if ( urlc != null )
{
   try
   {
      bri = new BufferedReader(new InputStreamReader(urlc.getInputStream()));
   }
   catch (UnknownServiceException e)
   {
      showHostCommError(1003, "A reading stream for host communication could not be created due to UnKnownServiceException.")
      showError(e);
      return;
   }
   catch (IOException e)
   {
      showHostCommError(1003, "A reading stream for host communication could not be created due to IOException.");
      showError(e);
      return;
   }
}
consoleDebug("reader created");

if (getMsgHandle() != null)
{
   getMsgHandle().changeText("Receiving data from host... Please Wait.");
   getMsgHandle().showIt();
}

// Receive data from host server (using Reader).
String msgOutLineString;
```

FIG. 7E

```
if ( bri != null )
{
   try
   {
      while ( (msgOutLineString = bri.readLine()) != null)
      {
         consoleDebug("msgOutLineString :" + msgOutLineString);

         msgOutString += msgOutLineString + "\n";
      }
   }
   catch (IOException e)
   {
      showHostCommError(1007, "Failure receiving data from host.");
      showError(e);
      return;
   }
}

consoleDebug("msgOutString :" + msgOutString);

consoleDebug("Through reading data from server");


if (getMsgHandle() != null)
{
   getMsgHandle().changeText("Closing host connection... Please Wait.");
   getMsgHandle().showIt();
}
```

FIG. 7F

```
// Cleanup
if ( bri != null )
{
   try
   {
      bri.close();
   }
   catch (IOException e)
   {
      showHostCommError(1008, "Input stream close error during host communication.");
      showError(e);
      return;
   }
}


// Check for and show the user any host communication ERRORS...
.
.
.


// Bring down any communication STATUS msgbox that we were
//    showing the user...
.
.
.

} /* end of stdio TalkToHome */
```

714

FIG. 7G

```
/**
 * This function shows debug info in Java Console of the
 *  browser only if TRACE flag has been turned on.
 *
 * Inputs to this function are:
 * 1. show - This is the string to display in the Java Console
 *
 */
public static void consoleDebug(String show)
{
   if (TRACE)
   {
      System.out.println("=>" + show);
   }
} /* end of consoleDebug */
```

714

FIG. 7H